CPE301 – SPRING 2020

MIDTERM 2

Student Name: Minsung Cho

Student #: 2001446442

Student Email: chom3@unlv.nevada.edu

Primary Github address: <https://github.com/cho-minsung>

Directory: <https://github.com/cho-minsung/midterm2>

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Atmega328PB FIT0450 Micro DC Motor with Encoder-SJ01 Potentiometer

Wires Breadboard TB6612FNG

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

if (Flag == 1)

{

PORTD |= (1<<DDD5);

PORTD &= ~(1<<DDD4);

*\_delay\_ms*(5000);

Flag = 1;

}

else

{

PORTD &= ~(1<<DDD5);

PORTD |= (1<<DDD4);

*\_delay\_ms*(5000);

Flag = 1;

}

CW and CCW code

ISR(TIMER1\_CAPT\_vect)

{

revTick = ICR1;

TCNT1 = 0;

revCtr++;

T10vs2 = 0;

}

ISR(TIMER1\_OVF\_vect)

{

t10vs2++;

}

int main()

{

while (1)

{

update\_ADC();

scale\_set\_speed();

Period = (*uint32\_t*)(revTick) + ((*uint32\_t*)T10vs2 \* 0x10000L);

calculate\_speed\_rpm();

USART\_SendString("RPM ");

*snprintf*(outs, sizeof(outs), "%f rpm ", (float)RPM);

USART\_SendString(outs);

USART\_SendString(" \r\n");

}

}

calculate 1x speed demo code

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

int main(void)

{

PORTC |= (1<<3); //pull up

DDRD |= 0xF0; //PORTD as Output

//CCW: AIN1 = L, AIN2 = H, PWM = H

//CW: AIN1 = H, AIN2 = L, PWM = H

while(1)

{

PORTD = 0x10; //counter clock wise

*\_delay\_ms*(5000);

PORTD = 0x00; //no movement

*\_delay\_ms*(5000);

PORTD = 0x20; //clock wise

*\_delay\_ms*(5000);

PORTD = 0x30; //no movement

*\_delay\_ms*(5000);

}

}

clock wise and counter clock wise DC motor code

#define *F\_CPU* 16000000UL

#define BAUD 9600

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <util/setbaud.h>

#include <stdio.h>

//given adc functions from previous assignments

void adc\_init();

void update\_ADC();

volatile float poten;

//given USART functions from previous assignments

void USART\_init();

void USART\_SendString(volatile char \*data);

//given Timer0 & 3 functions from previous assignments and lectures

void Timer0\_init();

void Timer3\_init();

//RPM floating point data functions

float RPM\_1x();

float RPM\_2x();

float RPM\_4x();

volatile float revTick1;

volatile float revTick4;

static char data[10];

int main(void)

{

//the port setup

DDRD |= (1<<6); //the speed control

DDRD &= ~(1<<4); //direction control

DDRD &= ~(1<<5);

PORTD &= ~(1<<4);

PORTD &= ~(1<<5);

DDRB |= (1<<0); //encoder speed capture for 1x and 2x

PORTB &= ~(1<<0);

DDRE |= (1<<2); //encoder speed capture for 4x

PORTE &= ~(1<<2);

DDRC |= (1<<0); //the potentiometer

PORTC &= ~(1<<0);

//initiation

adc\_init();

USART\_init();

Timer0\_init();

Timer3\_init();

sei();

while (1)

{

if (poten < 5)

{

USART\_SendString("1x encoding RPM: 0 \n");

//USART\_SendString("2x encoding RPM: 0 \n"); enable these for 2x or 4x

//USART\_SendString("4x encoding RPM: 0 \n");

}

else

{

USART\_SendString("1x encoding RPM: ");

*snprintf*(data, sizeof(data), "%f\n\r", RPM\_1x());

USART\_SendString(data);

USART\_SendString(" \n");

/\*

USART\_SendString("2x encoding RPM: ");

snprintf(data, sizeof(data), "%f\n\r", RPM\_2x);

USART\_SendString(data);

USART\_SendString(" \n");

\*/

/\*

USART\_SendString("4x encoding RPM: ");

snprintf(data, sizeof(data), "%f\n\r", RPM\_4x);

USART\_SendString(data);

USART\_SendString(" \n");

\*/

}

*\_delay\_ms*(1000);

}

}

void adc\_init(void)

{

ADMUX = (0<<REFS1) | (1<<REFS0) | (1<<ADLAR) | (0<<MUX2) | (0<<MUX1) | (0<<MUX0);

ADCSRA = (1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (0<<ADPS1) | (1<<ADPS0);

}

void update\_ADC(void)

{

unsigned char i = 5;

poten = 0;

while(i--)

{

ADCSRA |= (1<<ADSC);

while(ADCSRA & (1<<ADSC));

poten += ADC;

}

poten = poten/5;

}

void USART\_init(void)

{

UBRR0H = *UBRRH\_VALUE*;

UBRR0L = *UBRRL\_VALUE*;

UCSR0C = \_BV(UCSZ01)|\_BV(UCSZ00);

UCSR0B = \_BV(RXEN0)|\_BV(TXEN0);

}

void USART\_SendString(volatile char \*data)

{

while((\*data != '\0'))

{

while(!(UCSR0A & (1<<UDRE0)));

UDR0 = \*data;

data++;

}

}

void Timer0\_init(void)

{

TCCR0A |= (2 << COM1A0);

TCCR0A |= (0 << WGM10);

TCCR0B |= (0 << WGM12);

TIMSK0 |= (0x01);

TCNT0 = 0x00;

OCR0A = 0x00;

TCCR0B |= (0x02 << CS10);

}

void Timer3\_init(void)

{

TCCR3A = (0);

TCCR3B = (1 << WGM32);

OCR3A = (623);

}

float RPM\_1x()

{

float tempRPM1 = 0.0;

for (int i = 0; i < 20; i++)

{

TCNT3 = 0x00; //see the note for RPM4x for more info

TCNT4 = 0x00;

TIMSK4 |= (1 << 5);

TIMSK3 |= (1 << 1);

TCCR4B |= (0x07) | (1 << 6);

TCCR3B |= (0x04 << 0);

TCCR3B &= ~(0x07);

TCCR4B &= ~(0x07);

tempRPM1 = tempRPM1 + revTick4 \* 60 / 96;

}

return tempRPM1 / 20;

}

float RPM\_2x()

{

float tempRPM2 = 0.0;

for (int i = 0; i < 20; i++)

{

EICRA = 0x01; //see the note for RPM4x for more info

EIMSK = 0x01;

TIMSK4 = (1 << 5);

TIMSK3 = (1 << 1);

TCNT3 = 0x00;

TCNT4 = 0x00;

TCCR4B = (0x07) | (1 << 6);

TCCR3B = (0x04 << 0);

EIMSK = 0x00;

EICRA = 0x00;

TCCR3B &= ~(0x07);

TCCR4B &= ~(0x07 | (1 << 6));

tempRPM2 = tempRPM2 + revTick4 \* 60 / 192;

}

return tempRPM2 / 20;

}

float RPM\_4x()

{

float tempRPM4 = 0.0;

for (int i = 0; i <20; i++)

{

EICRA = 0x05;

EIMSK = 0x03; //Int1 is enabled

TIMSK1 = (1 << 5); //The input capture interrupts are enabled

TIMSK4 = (1 << 5);

TIMSK3 = (1 << 1); //The output capture interrupt is enabled

TCNT3 = 0x00; //Timer3 is reset

TCNT1 = 0x00; //Counter1 & 4 are reset

TCNT4 = 0x00;

TCCR1B = (0x07) | (1 << 6); //The counter1 & 4 are set as rising edges

TCCR4B = (0x07) | (1 << 6);

TCCR3B = (0x04); //The timer3 scaler is set

EIMSK = 0x00; //interrupts are reset

EICRA = 0x00;

TCCR3B &= ~(0x07); //clocks are cleared

TCCR1B &= ~(0x07 | (1 << 6));

TCCR4B &= ~(0x07 | (1 << 6));

tempRPM4 = tempRPM4 + (revTick1 + revTick4) \* 60 / 384;

}

return tempRPM4 / 20;

}

ISR(USART0\_TX\_vect)

{

//from the previous assignments

UCSR0B &= ~((1 << TXCIE0) | (1 << TXEN0));

}

ISR(ADC\_vect)

{

ADCSRA &= ~(1 << ADIE); //ADC interrupt is disabled

OCR0A = poten;

}

ISR(TIMER0\_OVF\_vect)

{

ADCSRA |= (1 << ADIE); //ADC interrupt is enabled

}

ISR(TIMER1\_CAPT\_vect)

{

//detects the inB

TIMSK1 &= ~(1<<5);

revTick1 = (ICR1);

}

ISR(TIMER3\_COMPA\_vect)

{

PORTE |= (1 << 0); //Timer1 & 4 capture interrupt is set

PORTB |= (1 << 0);

TIMSK3 &= ~(1 << 1);

}

ISR(TIMER4\_CAPT\_vect)

{

//detects the inA

TIMSK4 &= ~(1<<5);

revTick4 = (ICR4);

}

2x and 4x are commented out.

1. **SCHEMATICS**

- I used Timer0 PWM for controlling the speed of the motor to PD6

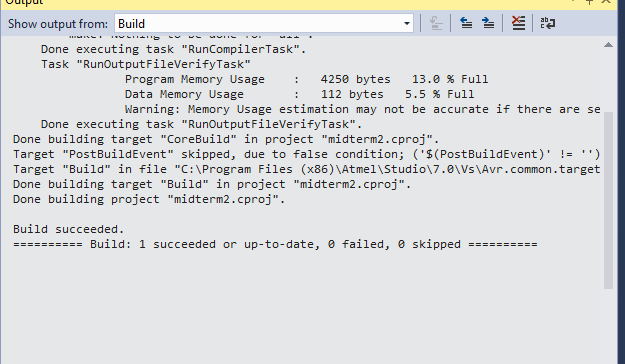
- I used pins PD5 and PD4 for direction control

- I used Timer1 ICP PB0 pin for encoder speed capture for 1x and 2x speed

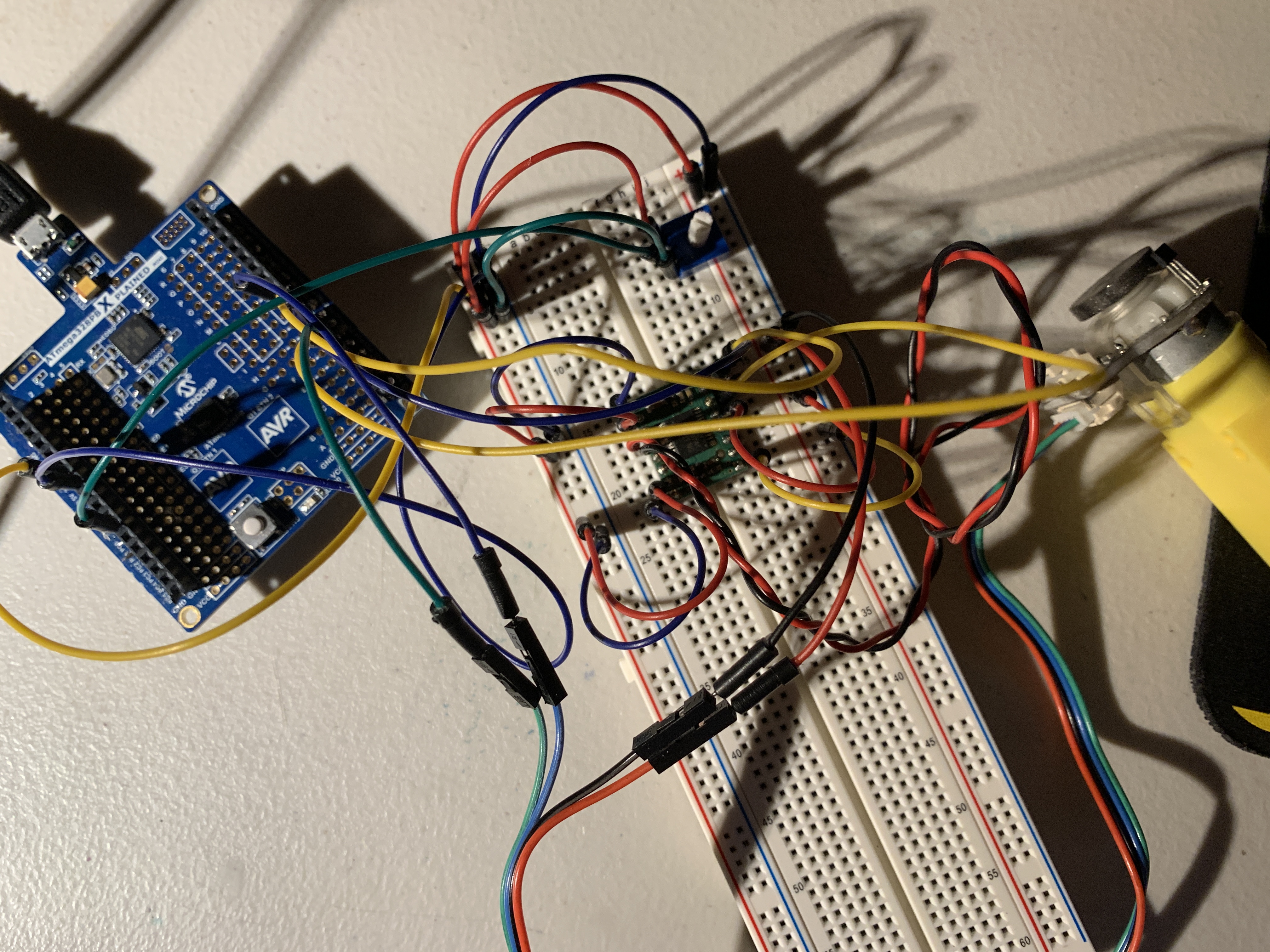
- I used Timer3 ICP PE2 pin for encoder speed capture for 4x speed

- I used PC0 for the potentiometer

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**



1. **VIDEO LINKS OF EACH DEMO**

<https://www.youtube.com/watch?v=295bseZTShs>

1. **GITHUB LINK OF THIS DA**

<https://github.com/cho-minsung/midterm2>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Minsung Cho